1. Explain CLR and its execution process

The Common Language Runtime (CLR) is a runtime environment provided by the .NET framework that manages the execution of .NET programs. It is responsible for executing managed code, providing services such as memory management, thread management, and exception handling.

The managed execution process includes the following steps, which are discussed in detail later in this topic:

1. Choosing a compiler.
   To obtain the benefits provided by the common language runtime, you must use one or more language compilers that target the runtime.
2. Compiling your code to MSIL.
   Compiling translates your source code into Microsoft intermediate language (MSIL) and generates the required metadata.
3. Compiling MSIL to native code.
   At execution time, a just-in-time (JIT) compiler translates the MSIL into native code. During this compilation, code must pass a verification process that examines the MSIL and metadata to find out whether the code can be determined to be type safe.
4. Running code.
   The common language runtime provides the infrastructure that enables execution to take place and services that can be used during execution.

2. What are C# variables and how to declare C# variables?

A variable is a named memory location that we can create in order to store data. Every variable has a type that determines what values can be stored in the variables.

●Declaration

[data type] [variable name];

● Initialization
[variable name] = [value];
● Combined
[data type] [variable name] = [value];

3. What are value types? Give some examples of predefined value types in C#.
In C#, value types are data types that represent a value directly. They are stored in memory as the value itself, rather than as a reference to an object stored elsewhere in memory. Examples of predefined value types in C# include:

- Integral types: These are types that represent integer values, such as int, short, long, and byte.
- Floating point types: These are types that represent real numbers, such as float and double.
- Decimal type: This is a type that represents a decimal value with a high degree of precision, suitable for financial and monetary calculations.
- Boolean type: This is a type that represents a logical value, either true or false.
- Char type: This is a type that represents a single Unicode character.
- Enumeration types: These are types that represent a set of named constants, such as enum.
- Struct types: These are types that represent a composite value, such as a struct.

Value types are often used to represent simple data values that do not require the overhead of a reference type. They are also often used when it is necessary to pass a value by value, rather than by reference.

4. Give examples for reference types and what is the difference between value type and

reference type.

● C# provides the following built-in reference types:

○ string

○ dynamic

○ object

● The following keywords are used to declare reference types:

○ class

○ delegate

○ interface

○ record

The main difference between value types and reference types is how they are stored in memory. **Value types are stored in memory as the value itself, while reference types are stored as a reference to an object stored elsewhere in memory.** This means that when a value type is passed as an argument to a method, a copy of the value is passed, whereas when a reference type is passed as an argument, a reference to the object is passed.

Another difference is that reference types are able to have a null value, whereas value types cannot. This means that reference types can represent the absence of a value, while value types cannot.

Finally, value types are generally more efficient to use than reference types, because they do not require the overhead of a reference. However, reference types are more flexible and can be used to represent more complex data structures.

5. What is immutable? Is string immutable or mutable?

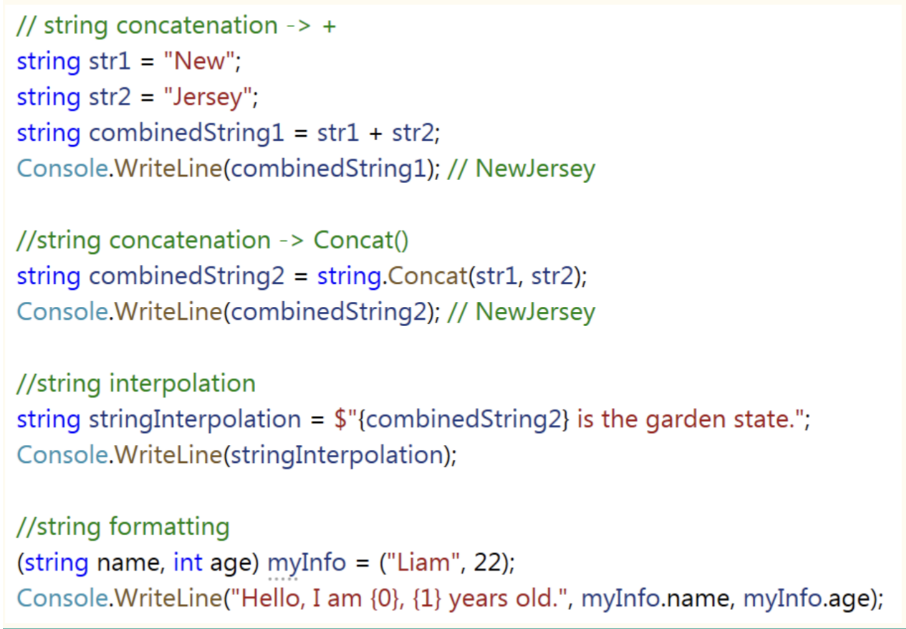
Immutable: they can't be changed after they've been created

String objects are immutable: When we assign a value to a string or modify the string, C# will create a new string.

6. What is the string intern pool in C# and why do we need the intern pool?
List some ways to Concatenate Strings In C#
The common language runtime(CLR) **conserves string storage** by maintaining a table, called the intern pool, that contains a single reference to each unique literal string declared or created programmatically in your program. Consequently, an instance of a literal string with a particular value only exists once in the system.

● For example, if you assign the same literal string to several variables, the runtime retrieves the same reference to the literal string from the intern pool and assigns it to each variable.

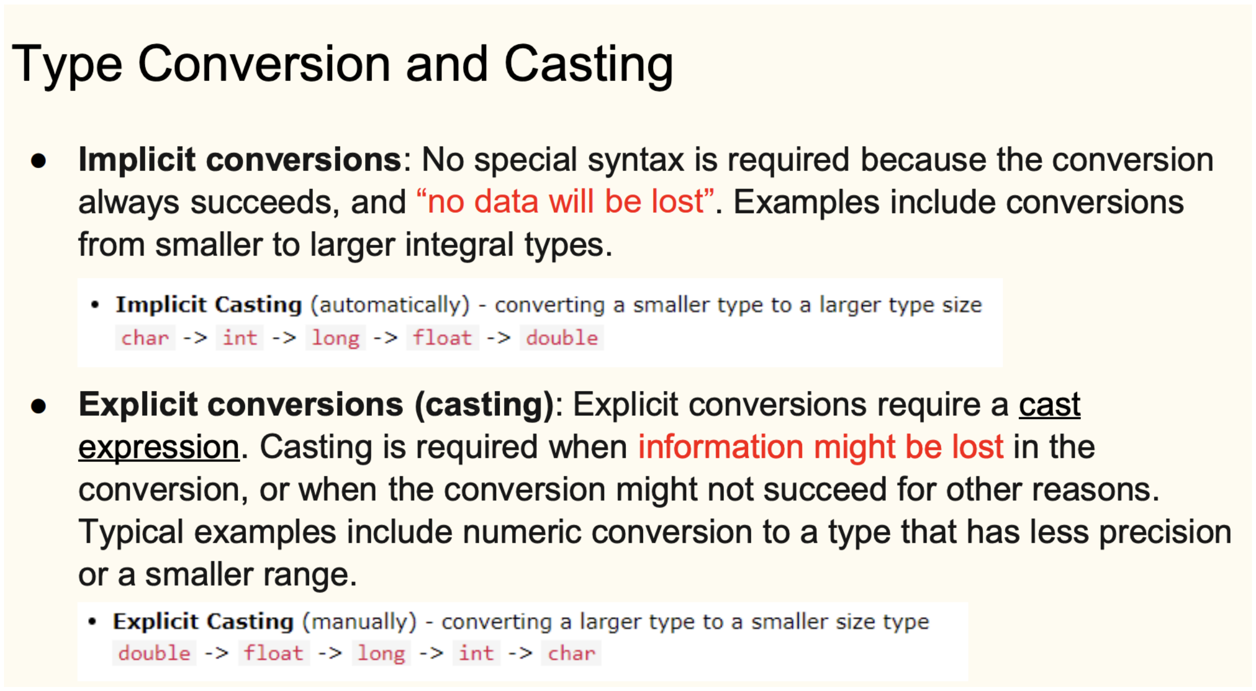
String Concatenation



```
// string concatenation -> +
string str1 = "New";
string str2 = "Jersey";
string combinedString1 = str1 + str2;
Console.WriteLine(combinedString1); // NewJersey

//string concatenation -> Concat()
string combinedString2 = string.Concat(str1, str2);
Console.WriteLine(combinedString2); // NewJersey

//string interpolation
string stringInterpolation = $"{combinedString2} is the garden state.";
Console.WriteLine(stringInterpolation);

//string formatting
(string name, int age) myInfo = ("Liam", 22);
Console.WriteLine("Hello, I am {0}, {1} years old.", myInfo.name, myInfo.age);
```

- string.Format();
- string.Join();
- Append() method in StringBuilder;

7. What is meant by casting a data type? What are the 2 kinds of data type conversions available in C#?



Type Conversion and Casting

- **Implicit conversions**: No special syntax is required because the conversion always succeeds, and "no data will be lost". Examples include conversions from smaller to larger integral types.
  - **Implicit Casting** (automatically) - converting a smaller type to a larger type size
    `char` -> `int` -> `long` -> `float` -> `double`
- **Explicit conversions (casting)**: Explicit conversions require a cast expression. Casting is required when information might be lost in the conversion, or when the conversion might not succeed for other reasons. Typical examples include numeric conversion to a type that has less precision or a smaller range.
  - **Explicit Casting** (manually) - converting a larger type to a smaller size type
    `double` -> `float` -> `long` -> `int` -> `char`

8. Will the following code compile? Why？

double myDouble = 11.11;

int myInt = myDouble;

No, there is no implicit conversion between these two types, so an explicit conversion is required. To convert a double value to an int value, you can use a type cast operator, such as (int), like this:

double myDouble = 11.11;

int myInt = (int)myDouble;

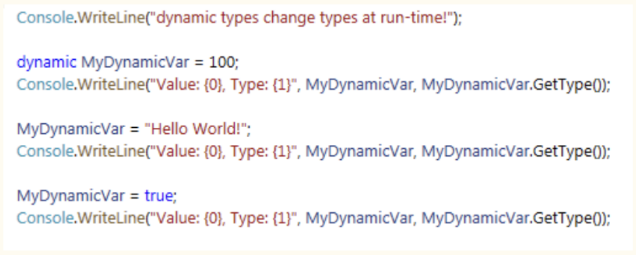
9. What is the dynamic type? How does it differ from var?

Dynamic type

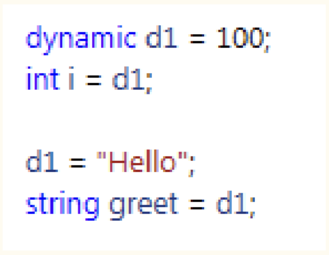
The **dynamic** type indicates that the use of the variable and references to its members **escapes compile-time type checking.** Instead, it resolves the type at run time.

The **dynamic** types change types at run-time based on the assigned value, so **the type dynamic only exists at compile-time, not run-time**.

The dynamic type variables are converted to other types implicitly.



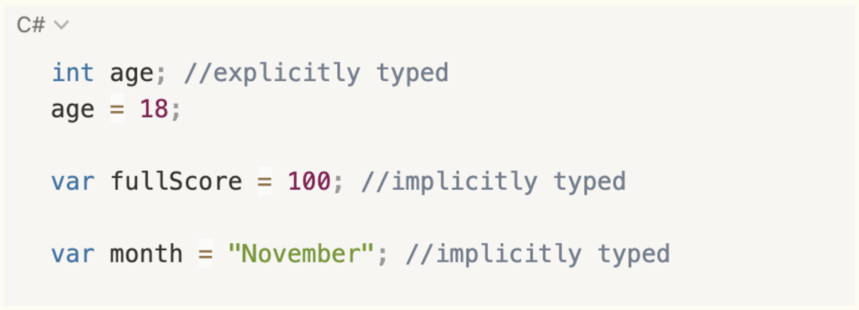
```
Console.WriteLine("dynamic types change types at run-time!");

dynamic MyDynamicVar = 100;
Console.WriteLine("Value: {0}, Type: {1}", MyDynamicVar, MyDynamicVar.GetType());

MyDynamicVar = "Hello World!";
Console.WriteLine("Value: {0}, Type: {1}", MyDynamicVar, MyDynamicVar.GetType());

MyDynamicVar = true;
Console.WriteLine("Value: {0}, Type: {1}", MyDynamicVar, MyDynamicVar.GetType());
```



```
dynamic d1 = 100;
int i = d1;

d1 = "Hello";
string greet = d1;
```

Implicit type - Var

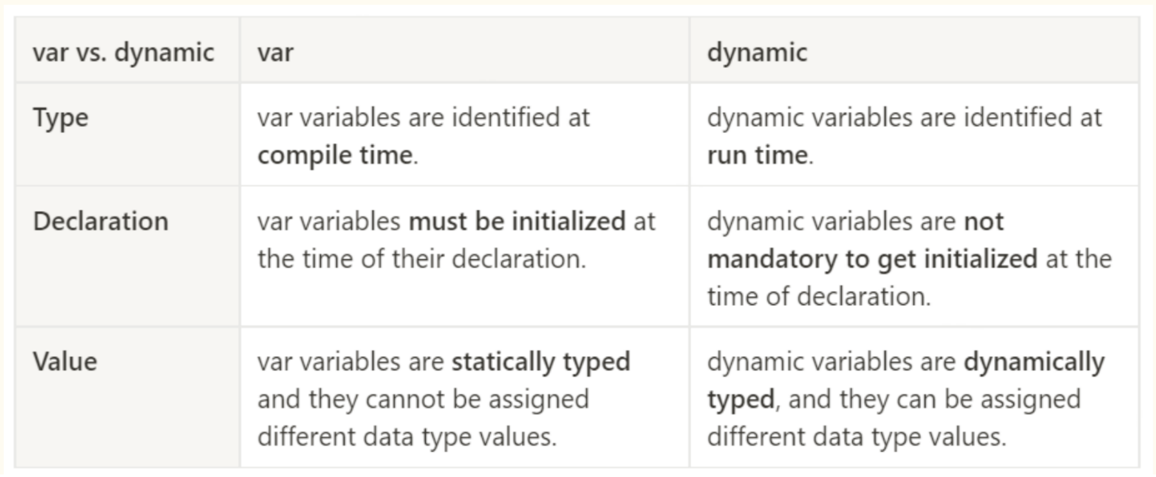
Beginning with C# 3, variables that are declared at method scope can have an implicit "type" **var**. An implicitly typed local variable is strongly typed just as if you had declared the type yourself, but the compiler determines the type.



C#

```
int age; //explicitly typed
age = 18;

var fullScore = 100; //implicitly typed

var month = "November"; //implicitly typed
```

Var vs. Dynamic

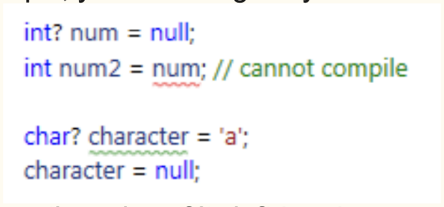


var vs. dynamic	var	dynamic
Type	var variables are identified at **compile time**.	dynamic variables are identified at **run time**.
Declaration	var variables **must be initialized** at the time of their declaration.	dynamic variables are **not mandatory to get initialized** at the time of declaration.
Value	var variables are **statically typed** and they cannot be assigned different data type values.	dynamic variables are **dynamically typed**, and they can be assigned different data type values.

10. What is the nullable value type?

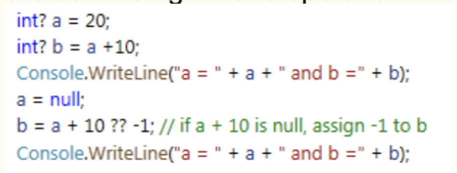
Nullable Value Type - ?

- Starting from C# 8.0, A nullable value type represents all values of its underlying value type and an additional null value. For example, you can assign any of the following three values to a ***bool?*** variable: **true**, **false**, or **null**.

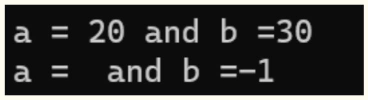


```
int? num = null;
int num2 = num; // cannot compile

char? character = 'a';
character = null;
```

- The null-coalescing operator ?? returns the value of its left-hand operand if it isn't null; otherwise, it evaluates the right-hand operand and returns its result. The ?? Operator doesn't evaluate its right-hand operand if the left-hand operand evaluates to non-null.



```
int? a = 20;
int? b = a +10;
Console.WriteLine("a = " + a + " and b =" + b);
a = null;
b = a + 10 ?? -1; // if a + 10 is null, assign -1 to b
Console.WriteLine("a = " + a + " and b =" + b);
```



```
a = 20 and b =30
a =  and b =-1
```

11. What are the results of the following expressions? Please include the calculation process. (logical operator)

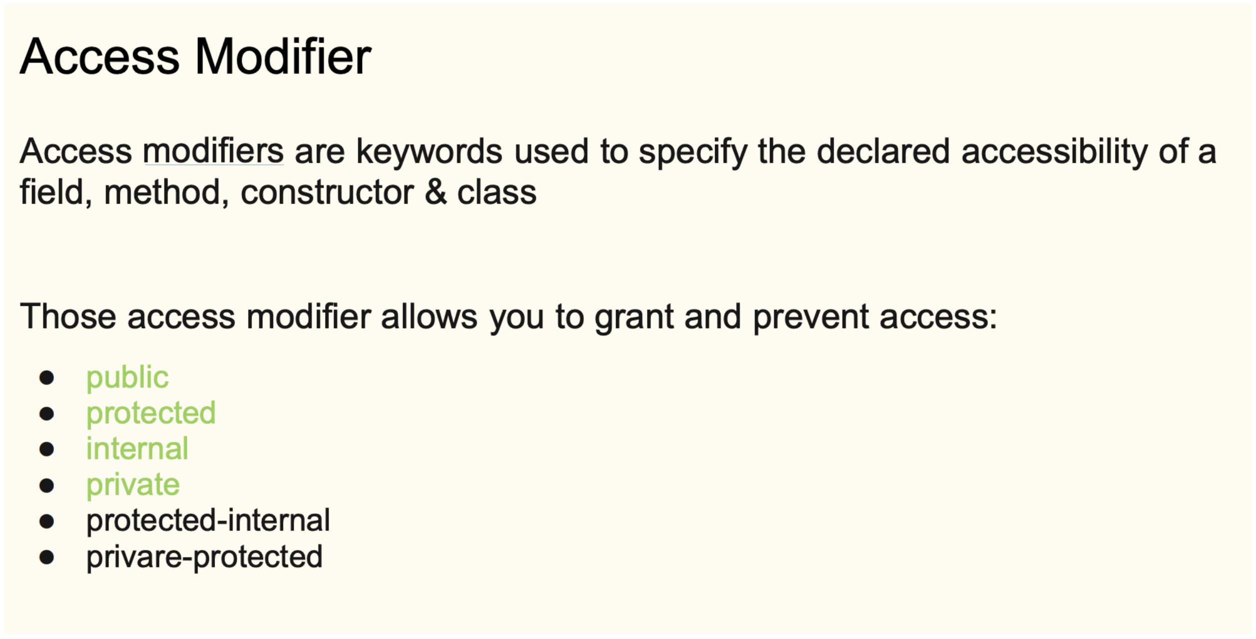
5: 0101

6: 0110

```
1) 5 & 6
 0101
& 0110
-------
  0100 = 4

2) 5 | 6
 0101
| 0110
-------
  0111 = 7

3) 5 ^ 6
  0101
^ 0110
-------
  0011  = 3
```

12. Why do we need to use the break statement in the Switch statement?
 It is used to terminate the execution of a case label or the default label in a switch statement. May cause a error.

13. What are access modifiers and their corresponding scopes in C#?

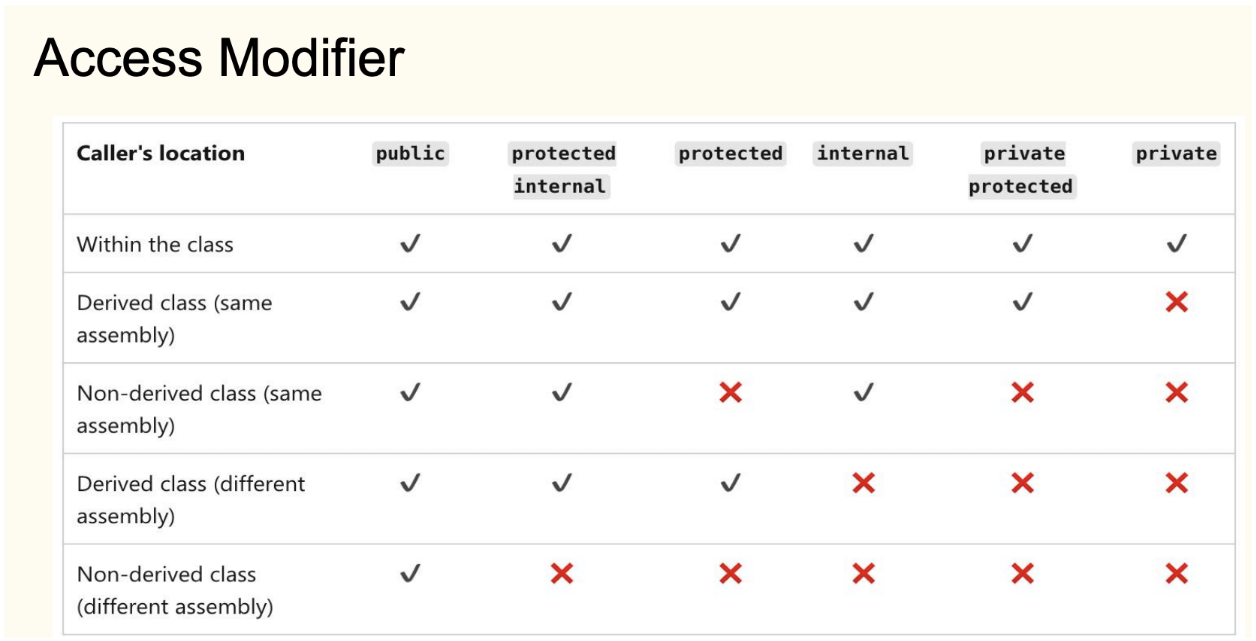


Access Modifier

Access modifiers are keywords used to specify the declared accessibility of a field, method, constructor & class

Those access modifier allows you to grant and prevent access:

- public
- protected
- internal
- private
- protected-internal
- privare-protected



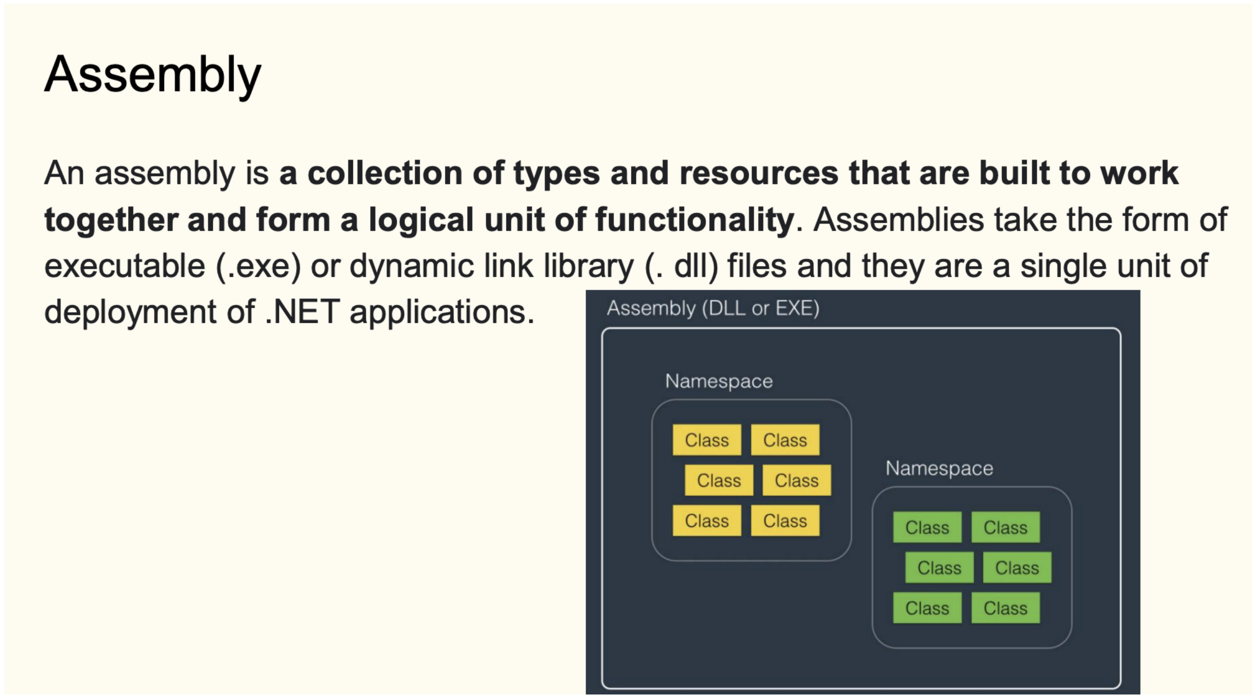
Access Modifier

Caller's location	`public`	`protected internal`	`protected`	`internal`	`private protected`	`private`
Within the class	✓	✓	✓	✓	✓	✓
Derived class (same assembly)	✓	✓	✓	✓	✓	✗
Non-derived class (same assembly)	✓	✓	✗	✓	✗	✗
Derived class (different assembly)	✓	✓	✓	✗	✗	✗
Non-derived class (different assembly)	✓	✗	✗	✗	✗	✗

14. Can you specify an access modifier for enumeration members?

No, it is not possible to specify an access modifier for enumeration members in C#. Enumeration members are always public and cannot be marked with an access modifier.

15. What is assembly?



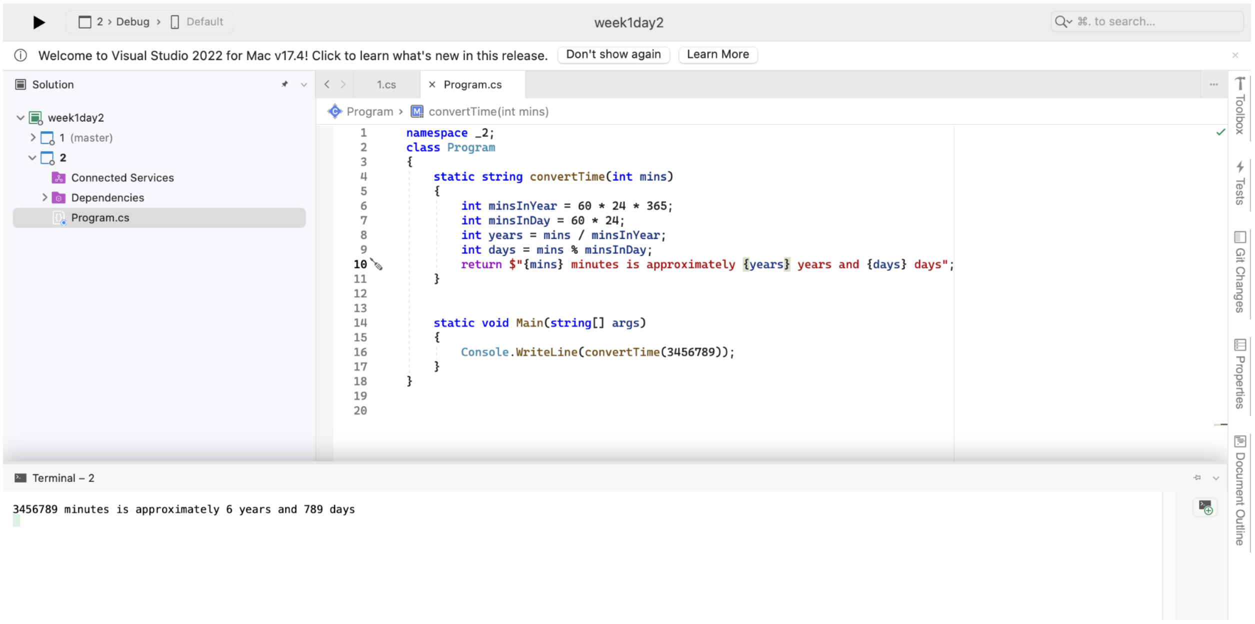
Assembly

An assembly is **a collection of types and resources that are built to work together and form a logical unit of functionality**. Assemblies take the form of executable (.exe) or dynamic link library (. dll) files and they are a single unit of deployment of .NET applications.

Coding

1.

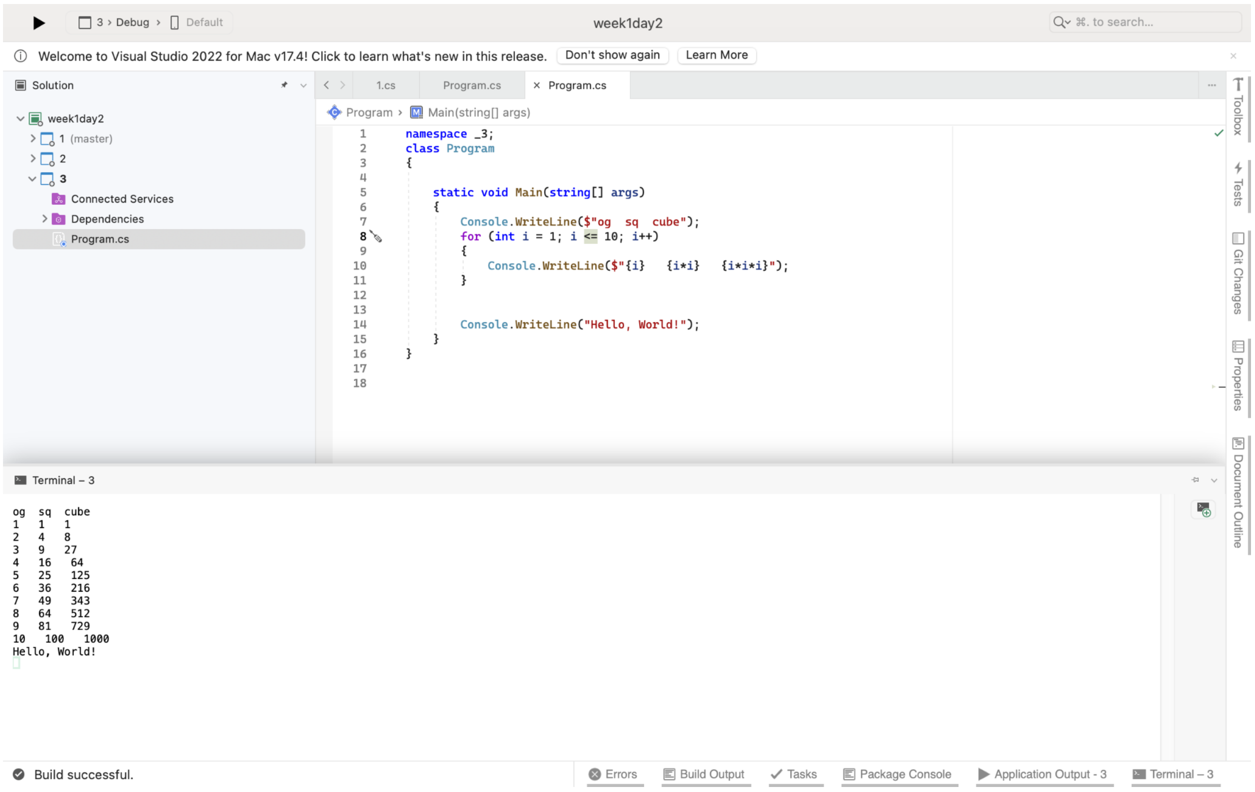
2.



```
namespace _2;
class Program
{
    static string convertTime(int mins)
    {
        int minsInYear = 60 * 24 * 365;
        int minsInDay = 60 * 24;
        int years = mins / minsInYear;
        int days = mins % minsInDay;
        return $"{mins} minutes is approximately {years} years and {days} days";
    }


    static void Main(string[] args)
    {
        Console.WriteLine(convertTime(3456789));
    }
}
```

```
3456789 minutes is approximately 6 years and 789 days
```

3.

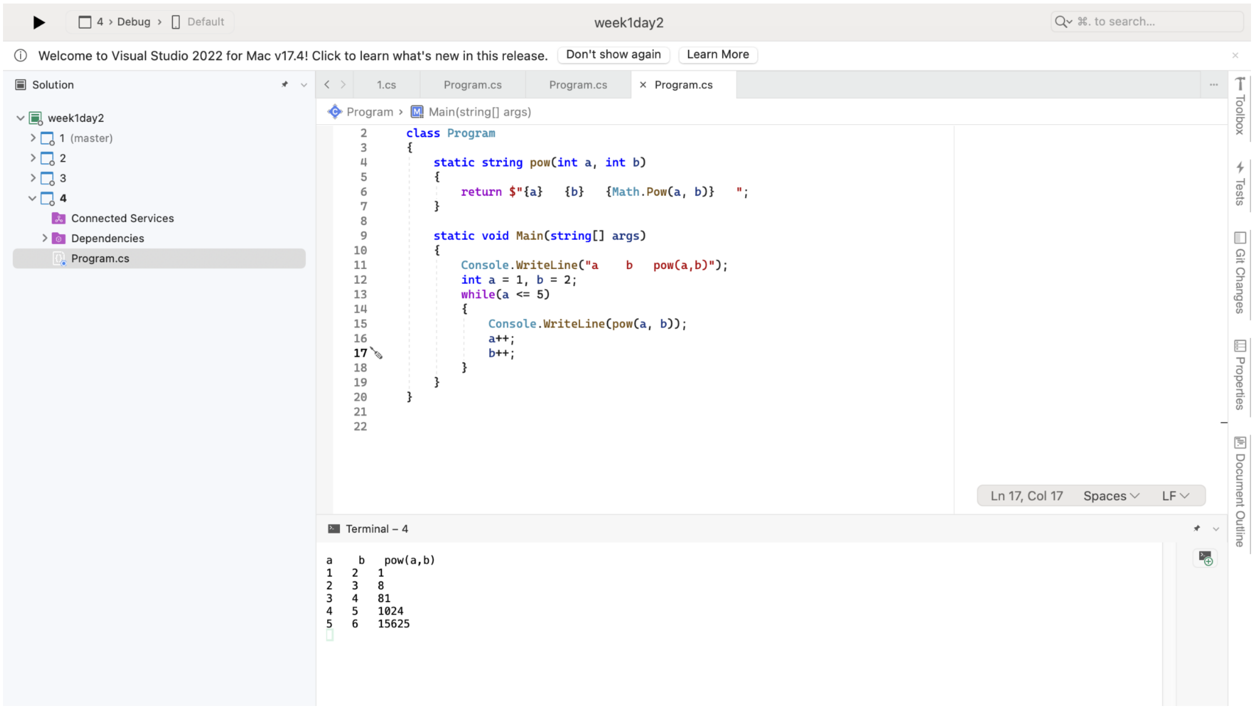


```
namespace _3;
class Program
{

    static void Main(string[] args)
    {
        Console.WriteLine($"og  sq  cube");
        for (int i = 1; i <= 10; i++)
        {
            Console.WriteLine($"{i}   {i*i}   {i*i*i}");
        }


        Console.WriteLine("Hello, World!");
    }
}
```

```
og  sq  cube
1   1   1
2   4   8
3   9   27
4   16   64
5   25   125
6   36   216
7   49   343
8   64   512
9   81   729
10   100   1000
Hello, World!
```

Build successful.

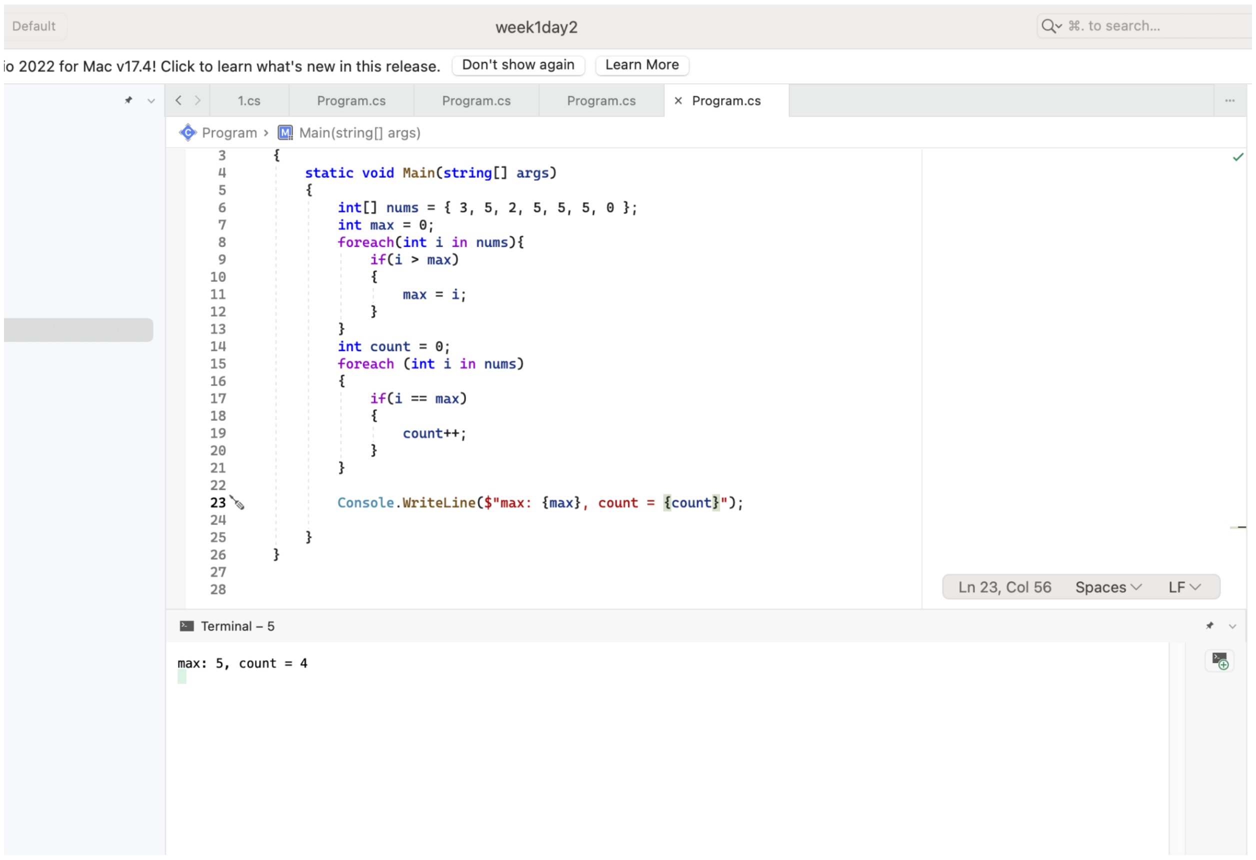
4.



```
class Program
{
    static string pow(int a, int b)
    {
        return $"{a}    {b}    {Math.Pow(a, b)}    ";
    }

    static void Main(string[] args)
    {
        Console.WriteLine("a    b    pow(a,b)");
        int a = 1, b = 2;
        while(a <= 5)
        {
            Console.WriteLine(pow(a, b));
            a++;
            b++;
        }
    }
}
```

```
a    b   pow(a,b)
1   2   1
2   3   8
3   4   81
4   5   1024
5   6   15625
```

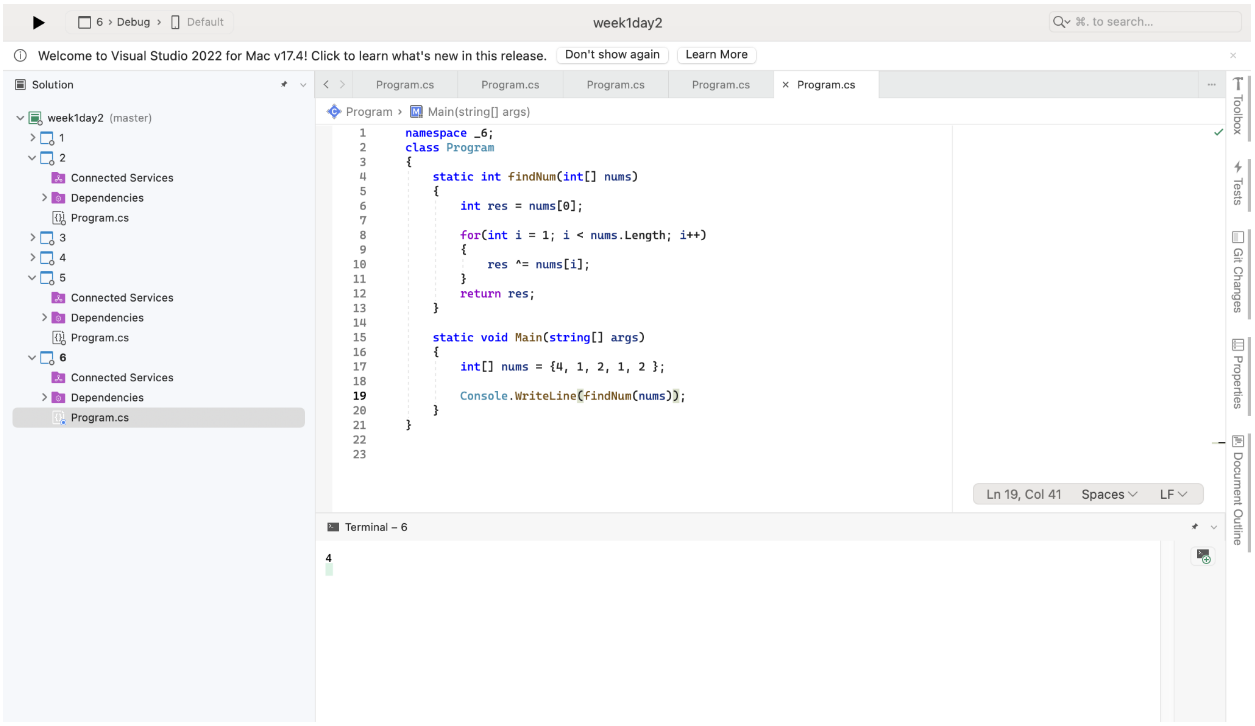
5.



```csharp
    {
        static void Main(string[] args)
        {
            int[] nums = { 3, 5, 2, 5, 5, 5, 0 };
            int max = 0;
            foreach(int i in nums){
                if(i > max)
                {
                    max = i;
                }
            }
            int count = 0;
            foreach (int i in nums)
            {
                if(i == max)
                {
                    count++;
                }
            }

            Console.WriteLine($"max: {max}, count = {count}");

        }
    }
```

```
max: 5, count = 4
```

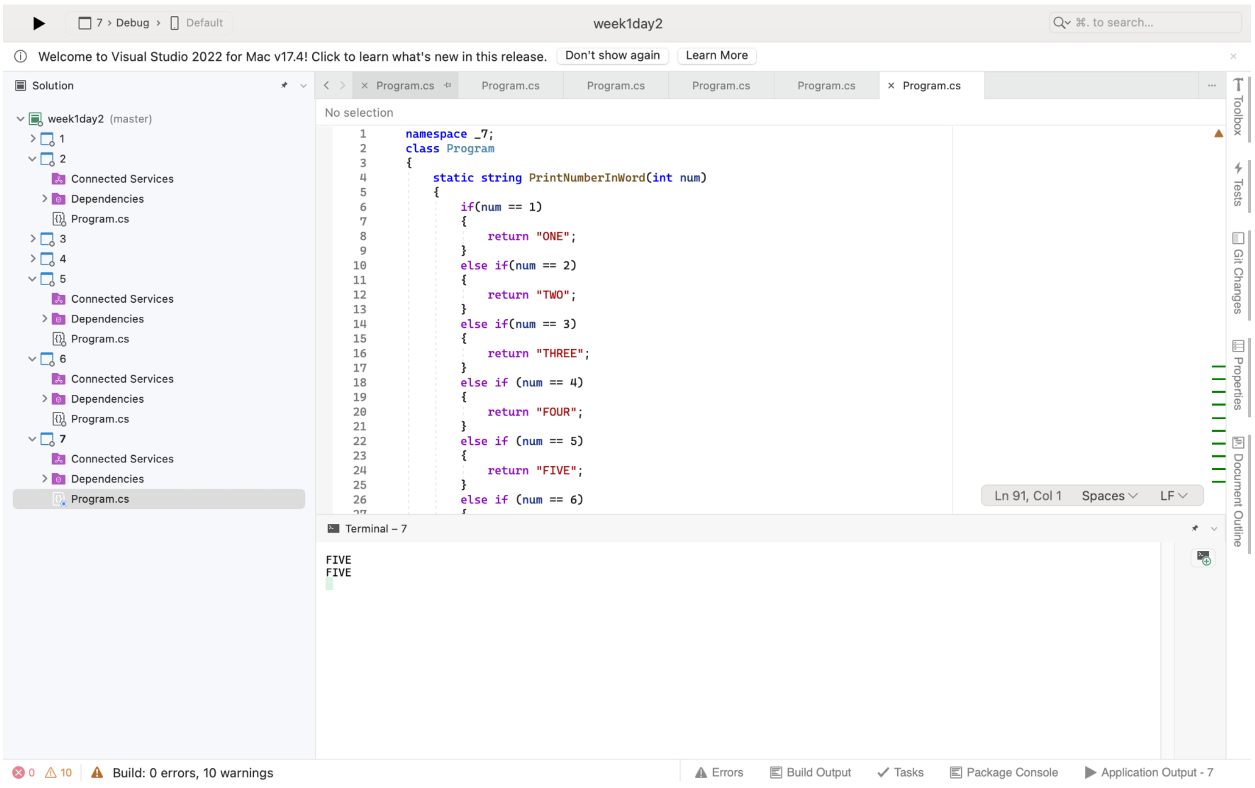
6.



```
namespace _6;
class Program
{
    static int findNum(int[] nums)
    {
        int res = nums[0];

        for(int i = 1; i < nums.Length; i++)
        {
            res ^= nums[i];
        }
        return res;
    }

    static void Main(string[] args)
    {
        int[] nums = {4, 1, 2, 1, 2 };

        Console.WriteLine(findNum(nums));
    }
}
```

```
4
```

7.

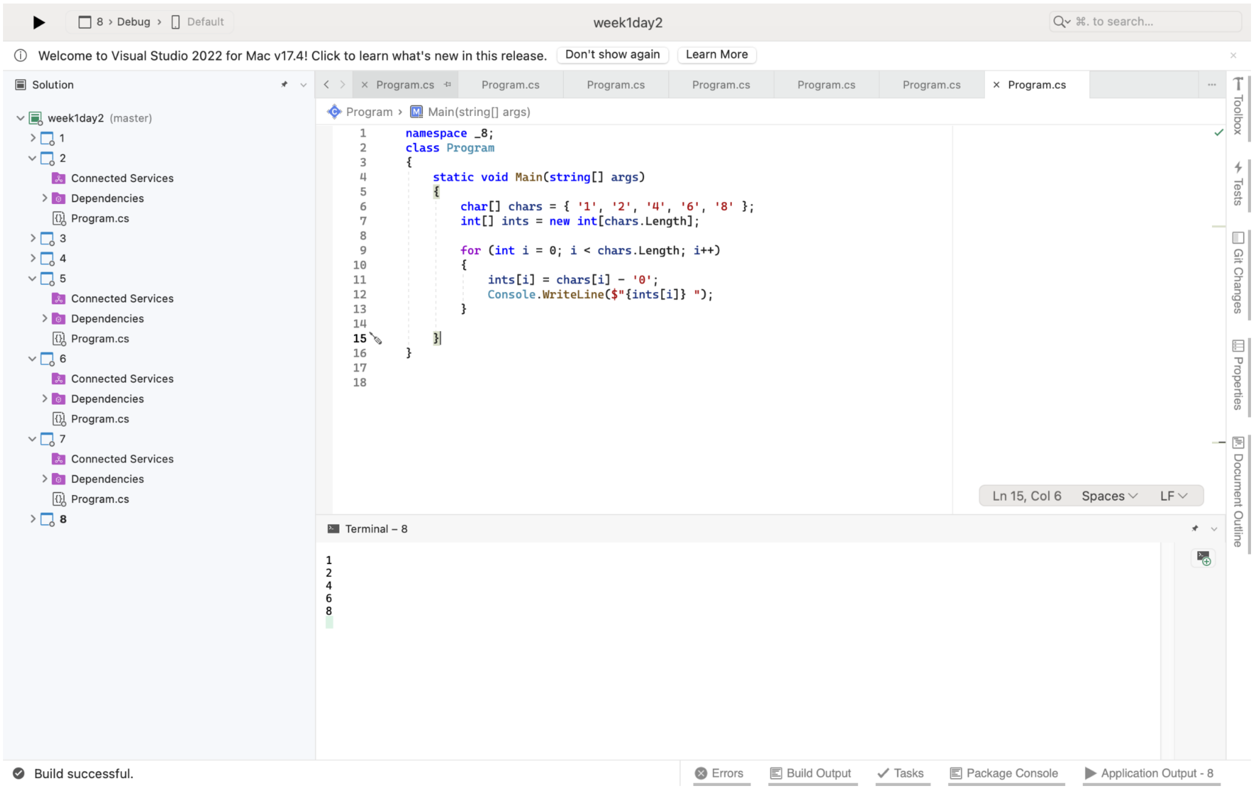


```
namespace _7;
class Program
{
    static string PrintNumberInWord(int num)
    {
        if(num == 1)
        {
            return "ONE";
        }
        else if(num == 2)
        {
            return "TWO";
        }
        else if(num == 3)
        {
            return "THREE";
        }
        else if (num == 4)
        {
            return "FOUR";
        }
        else if (num == 5)
        {
            return "FIVE";
        }
        else if (num == 6)
```

```
FIVE
FIVE
```

Build: 0 errors, 10 warnings

8.



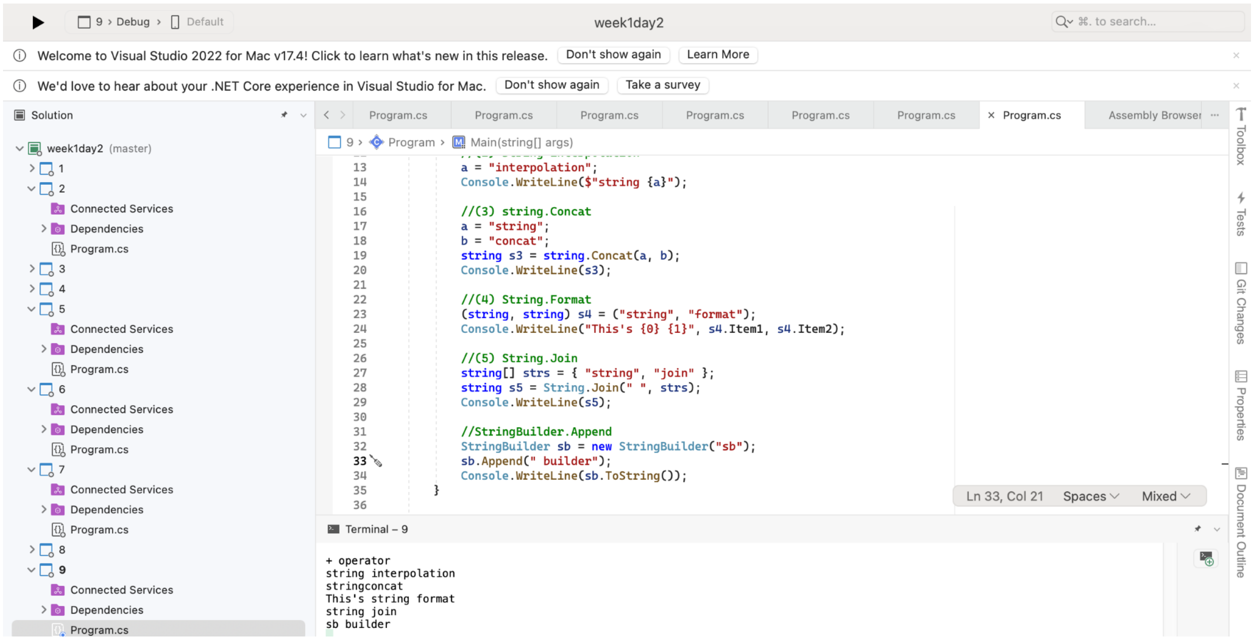
Welcome to Visual Studio 2022 for Mac v17.4! Click to learn what's new in this release. Don't show again Learn More

```
namespace _8;
class Program
{
    static void Main(string[] args)
    {
        char[] chars = { '1', '2', '4', '6', '8' };
        int[] ints = new int[chars.Length];

        for (int i = 0; i < chars.Length; i++)
        {
            ints[i] = chars[i] - '0';
            Console.WriteLine($"{ints[i]} ");
        }

    }
}
```

```
1
2
4
6
8
```

Build successful.

9.



Welcome to Visual Studio 2022 for Mac v17.4! Click to learn what's new in this release. Don't show again Learn More

We'd love to hear about your .NET Core experience in Visual Studio for Mac. Don't show again Take a survey

```
        a = "interpolation";
        Console.WriteLine($"string {a}");

        //(3) string.Concat
        a = "string";
        b = "concat";
        string s3 = string.Concat(a, b);
        Console.WriteLine(s3);

        //(4) String.Format
        (string, string) s4 = ("string", "format");
        Console.WriteLine("This's {0} {1}", s4.Item1, s4.Item2);

        //(5) String.Join
        string[] strs = { "string", "join" };
        string s5 = String.Join(" ", strs);
        Console.WriteLine(s5);

        //StringBuilder.Append
        StringBuilder sb = new StringBuilder("sb");
        sb.Append(" builder");
        Console.WriteLine(sb.ToString());
    }
```

```
+ operator
string interpolation
stringconcat
This's string format
string join
sb builder
```